

第3章 システムを監視する組み込みソフトウェア設計

舘 伸幸

メモリもペリフェラルも信用しない，それもこれも安全のため

組み込みソフトウェアは，プロセッサやメモリなどから構成されるシステム上に搭載されて動作している．このとき，プログラム自体に不具合があれば，当然システムにも不具合が発生する．また，プログラム自体に不具合がなくても，ノイズなどの外的要因によって，メモリに書き込んだはずのデータが変わってしまうことがあり得る．ここでは，そのような想定外の事態にできるだけ対応するために，ソフトウェアが自分自身やハードウェアを監視して不具合を検出するための設計について解説する．（編集部）

ソフトウェア技術者は，ハードウェアに対して絶対的な信頼を期待しがちです．メモリに書き込んだデータの値は電源を切らない限り変わらず，ペリフェラルの設定は意図的に変えない限り変わらない，と．ソフトウェア部品に対してさえも，不具合さえなければ，いつも必ず正しく動作すると信じています．

しかし「不具合がない」とは，ほとんどの場合「想定された要件に対しては」という条件が付きます．世の中の事故事例を解析するまでもなく，身近なクレームを見れば分かるように，綿密なテストのすきを突くような問題がしばしば発生しています．

このような問題に対しては，これまでは「考慮不足などによる対象モジュールの不具合」としてとらえ，対策してきました．もちろんそのことは重要ですが，十分にテストされた部品でも特定の条件においては期待通り動作しないことがあり得る，つまり「部品は（ソフトウェアでもハードウェアでも）故障することがある」という発想の下で対策を作り込んでおくという，機能安全の考え方も重要です．

従来の手法を継続・発展させて信頼性を高めつつ，故障を未然に防いだり，故障してもそれを検出して回復動作を行うことで，システムとしての異常動作を最小限に抑えたりすることが求められます．

ここでは，ソフトウェア（プログラム）で異常を検出するための監視方法について解説します．監視対象は大きく二つに分けられます．一つはソフトウェアの自己監視であり，もう一つはマイコンやセンサなどの物理層の監視です[(1)][注1]．

1．プログラムを自己監視する

組み込みソフトウェアの最も単純な階層構造の例を，**図1**に示します．ここでアプリケーション・ソフトウェアに，自分自身の動作状態と，下位層であるミドルウェア（汎用サブルーチン）やデバイス・ドライバの動作状態を監視する機能を持たせます．同じように，デバイス・ドライバには，自身と下位層であるハードウェアを監視する機能を持たせます．

● 処理の実行順序を監視する

まずは，シーケンシャルな処理を実施するアプリケーション・ソフトウェアについて，順番通り処理を実行できているかどうかを監視するプログラムを考えてみます（**図2**）．

注1：異常検出のための監視対象については，IEC 61508規格の第3部 7.2.2.9に記述されている．なお第3部の冒頭には，第1部と第2部を理解した上でのみ用いるようにとの指示がある．

KeyWord　機能安全，IEC 61508，信頼性，監視，予防，耐タンパ，チェック・サム

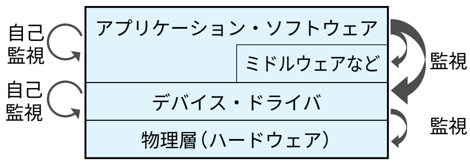


図1　組み込みシステムの階層構造
ここに示したのは，最も単純な階層構造の例である．シンプルなシステムでも，階層構造を明確にしておくことは重要．信頼性だけでなく，安全対策としても有効である．

処理モジュールB，C，D，Eは，主処理モジュールAから順番に呼び出されて実行される仕様とします．

B～Eの各処理モジュールにおいては，主処理モジュールであるAから呼び出されているか，直前の処理モジュールが適正なものであるかをチェックします．主処理モジュールAにおいては，処理が1周するごとに実行順序が適正であったかをチェックします．

実装としては，各モジュールがRAM上に実行履歴を書き込んでいき，その値を別モジュールがチェックするといった方法が考えられます．

図3にもう少し具体的な例を示します．主処理モジュール（モジュールA）は各処理モジュール（モジュールB～E）を呼び出すとき，u1MOD_Aという値を引き数に渡します．

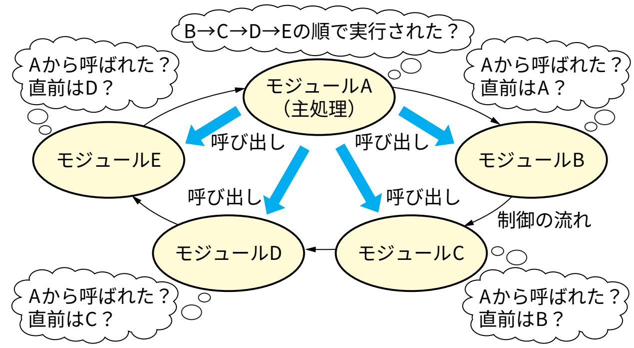


図2　通り道の監視
シーケンシャルな処理システムの場合，自身の実行経路は容易に記録・確認できる．

呼び出されたモジュールは引き数がu1MOD_Aであることを調べることで，正しく主処理モジュールから呼び出されたことを確認します（**図3**の①）．引き数は通常スタックまたはレジスタで渡されますが，暴走など異常動作でたまたまこの処理に制御が移った場合，引き数が正しい値と一致する確率は低いという原理を利用しています．なお，この例ではu1という1バイトで記述していますが，u4（unsigned long）など値を大きくすることで異常呼び出しの検出精度を上げることもできます．

各処理モジュールは呼び出し元が正しいことを確認した

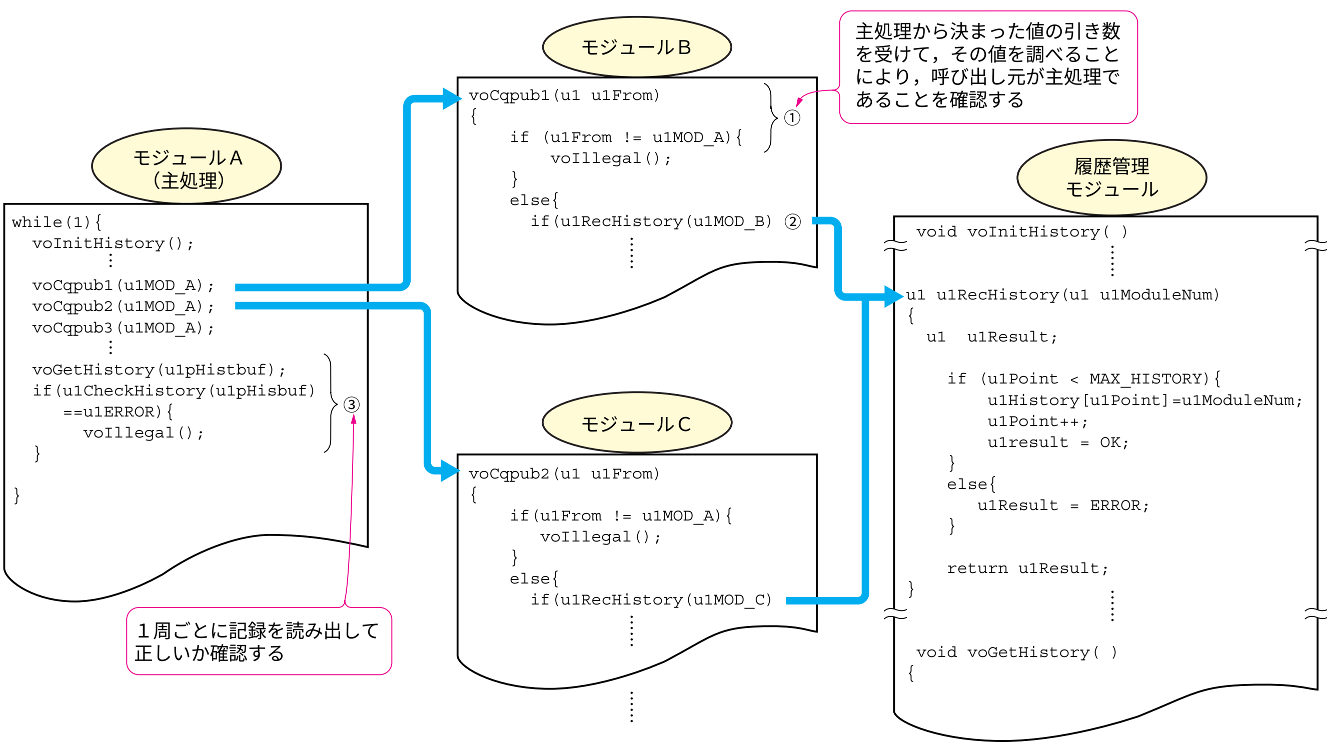


図3　シーケンシャル処理における監視の実装例
履歴の保存や読み出しは専用のモジュールで行う．安易にグローバル変数で実装したりすると，品質の低下につながる．

ら，次に自分の実行履歴を残します(図3の②)．そして，各処理モジュールの処理が一巡した時点で主処理モジュールが実行履歴を読み出し，正しい順に実行されたかどうかを照合します(図3の③)．

● 処理のタイミングを監視する

今度は，一定時間ごとに処理が呼び出される場合の例を図4に示します．処理を開始する時点で，正しいタイミングで呼び出されているか，また前回呼び出されてからの経過時間は適正かをチェックします．

さらに，スケジューラ方式でなくても，処理の終了時点で処理自体にかかった時間をチェックし，システムの異常を検知することもできます．

いずれにしても，実装においては専用のタイマ・サービス(ハードウェア・タイマ)を用意する必要があるでしょう．

● 下位層を監視する

下位層を監視する例として，アプリケーション・ソフトウェアからデバイス・ドライバを監視する例を示します(図5)．アプリケーション・ソフトウェアはある時点でデバイス・ドライバを呼び出します．同時にタイマを設定し，一定時間以上応答がない場合(タイムアウトした場合)は障害があったとみなして，デバイス・ドライバに対して初期化などの復帰処理を行います．

デバイス・ドライバからデバイス(物理層)にアクセスする場合も，同じ手法が適用できます．

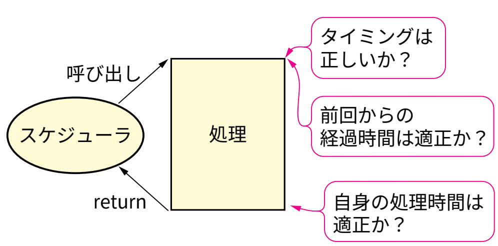


図4　一定時間ごとに呼び出される処理における監視
必ず一定間隔で実行される処理群では，このような監視も有効である．

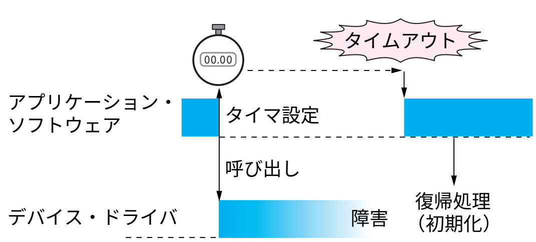


図5　下位層の監視
テストされた部品であっても，すべての外乱条件を網羅していることはあり得ない．障害は起こり得るものとして使う．

● 一つの要件に対して複数の処理を用意する

そのほか，IEC 61508にはダイバース・プログラミング(diverse programming)という言葉が登場します．聞きなれない言葉だと思うので，紹介します．

これは，同じ要件処理に対して異なる処理方法を複数用意し，実行する方法です(図6)．理想的には，異なるハードウェア(CPU)にそれぞれ異なるアルゴリズムで実装するのが望ましいようですが，同じCPUにアルゴリズムだけ異なる処理を載せることでも効果を得られると思います．この方法では，処理結果の誤差許容範囲と，範囲逸脱時の処理仕様の設計がポイントになるでしょう．

2. 物理層を監視する

物理層が正常に動作しているかどうかを監視する方法は，対象回路の動作仕様ごとに異なります．

ソフトウェアの動作と非同期に値や状態が変化するようなものや，CPUそのもの(実行ユニット)の異常などについては，ソフトウェアから異常を検出するのは困難です．それでも，一般的に適用できる技法はいくつかあります．

● メモリやレジスタの内容を疑う

まずは自分自身，つまりROMコードのチェック・サム

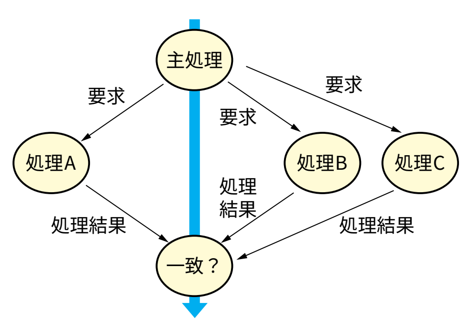


図6　ダイバース・プログラミング
同一の処理を複数の異なるアルゴリズムで処理し，一致することを確認する方法．これにより，特定の処理における特定条件での不具合を検出することができる．

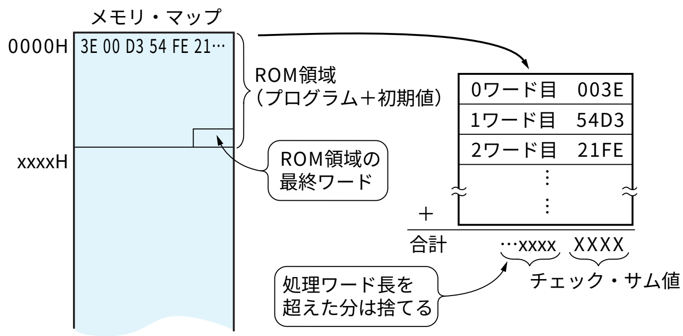


図7 ROMチェック・サム

ROM領域の内容の総和を求めてチェックする方法．通常は1～4バイトの単位を処理長として総和を計算し，下位の処理長分をチェック・サム値とする(けたあふれ分は捨てる)．ROM領域における最終ワードの直前までのチェック・サムを計算し，最終ワードにはその値の2の補数を入れておくという方法がよくとられる．チェック時には最終ワードまで計算して，チェック・サム値がゼロになればOKとする．故意でない破壊に対しては簡単で効果的な方法であるが，不正改造に対する防御(耐タンパ)目的なら，計算方法や処理範囲などに工夫が必要である．

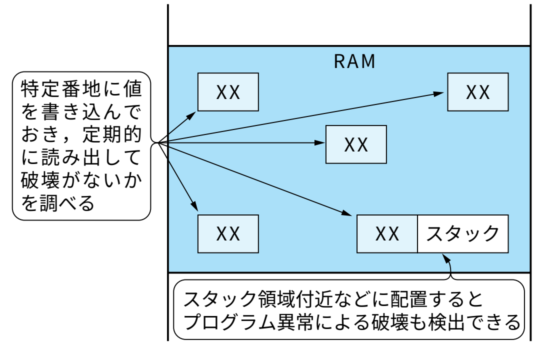


図8 RAMの破壊監視

特に外部に接続されたRAMはノイズによる影響を受けやすいので，より厳しい監視や未然防止技法を適用しておく．

を確認します(図7)．チェック・サムにより，製造工程での書き込みミスや劣化，動作異常によるROM破壊，第三者による改変などを検出できます．通常はシステム起動時に1回チェックするという仕様が多いのですが，長時間連続稼働するようなものは定期的にチェックを行うとよいでしょう．改変に対する防御を考慮する場合は，(チェック・ルーチンをスキップさせることへの対抗として)複数個所でチェックを実施すると効果的です．

ROMと同様に，RAMの異常にも対処が必要です．ノイズなどによる内容破壊を監視しなければなりません．RAMそのものにハードウェア的な誤り検出機構(パリティ・ビットなど)がある場合は，積極的にそれを利用します．ない場合は，例えばRAMの特定番地にユニークな値を書き込んでおき，それを定期的に調べるという方法もあります(図8)．これはRAMの物理的異常だけでなく，プログラムの誤動作による変数(領域)破壊の検出にも有効です．

記録データやパラメータなど，アクセスがそれほど頻繁でないものは，データそのものにチェック・サムを付けたりデータを異なるアドレスに2重に保存したりします注2．特に外部バスに接続されたメモリ領域は，ノイズなどによる誤アクセスの発生が考えられるので，必ずなんらかの対策を講じておく必要があります．

マイコンの内蔵ペリフェラルのレジスタについても，データ破壊やデータ化けへの考慮が必要です．ペリフェラル動作の許す範囲で，特に動作モードの設定値などに定期的に正しい値を上書きするといった対策が有効です．これは監視というよりも，予防措置といえます注3．

● 安全機能をわざと起動させてみる

なんらかの安全機能をわざと起動させて動作をチェック

注2：重複保存したデータに不一致が検出された場合は，一般的にはそれらを破棄して初期化を行う．

注3：設定値を読み出して異常だったら正常値を書き込むよりも，いきなり正常値を上書きする方が手っ取り早い．

COLUMN 要件と設計の関係

設計において要件を網羅することは当然ですが，逆に要件にないものは決して実装してはいけません．安全対策としていろいろな技術がありますが，それらはすべて要件として定義されているべきです．すなわち，システム設計者がシステムの安全に関するアーキテクチャの一環として，ソフトウェアに対する要件を把握し，定義する必要があります．そうでないと，せっかく入れた対策がシステム全体との不適合を起こしてしまい，それ自体が不具合の原因となりかねないからです．

しかし，システム設計者がソフトウェアの詳細設計レベルのテクニックに精通しているとは限りません．ソフトウェア設計者は，要件に対して受け身にならず，安全の観点から積極的にソフトウェアでできる対策案を提案し，要件にフィードバックしていくべきでしょう．

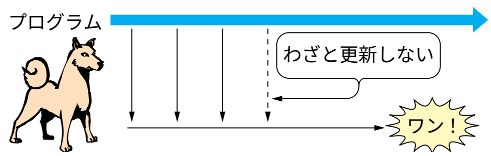


図9　ウォッチドッグ・タイマ
定期的にあるタイミングで更新を停止し，正常に機能するかどうかを検査する．

する方法もあります．図9にウォッチドッグ・タイマの例を示します．ウォッチドッグ・タイマを使うこと自体はポピュラーな方法ですが，ここではウォッチドッグ・タイマが故障していないことを確認するために，定期的にわざと更新処理をやめることで，ウォッチドッグ・タイマが正常に働くかどうかをチェックしています．このように，システムの動作中に，安全機能を定期的にチェックすることが有効です．

ここまで説明してきた方法は，特に目新しいものではありません．しかし，それゆえに要件として明記されていなかったり，設計現場では一種の常識としてぞんざいに扱われている場合もあります．安全上のどのような問題に対してどのような技術があるのか，そしてどれを実装するのかを明示し，設計段階で十分レビューすることが大切です．

なお，機能安全は局所的にテクニックを投入するだけで実現できるものではありません．システムとして安全を実現するために，設計工程全体で取り組んでいく必要があります（p.37のコラム「要件と設計の関係」を参照）．

3．異常を検出したときの対処

監視処理で問題が検出されたとき，どのように対処すればよいでしょうか．IEC 61508の中では，第3部の付属書Aにいくつか技法が記述されています．

1）フォワード・リカバリとバックワード・リカバリ

どちらも元はデータベースの障害復旧の方法です．なんらかの障害が検出されたときに，記録されている正常だったときのデータをもとに復旧を図ります．

組み込みソフトウェアでは，リセット直後の初期状態や，もし実行履歴を記録しているシステムであれば障害発生直前の状態を基に復旧を行います．基本はリトライとスキップです．障害が検出されたときは，一つ前の処理に戻るか，あるいはリセットにより初期状態に戻し，再度実行を試みます（リトライ）．しかし，物理障害など恒久的な障害の場合，無限ループに陥る危険もあります．必ず回数制限をかけるなどの対策が必要です．

回復不可能な障害が検出された場合は，該当処理をスキップすることで，とにかく先へ進める方法もあります．システムの停止が安全を損なうような場合，この方法は有効です．

2）グレイスフル・デグラデーション

直訳すれば「しとやかな規模縮小」でしょうか．システムが危機的状態であるとき，優先度の低いものから順に処理を切り離して，なんとか生き延びようとさせる方法です．通信ドライバなどで，通信レートを落としてでもなんとか通信を継続するといった方法もこれにあたります．

● 改造から守ることも一つの機能安全

不正改造による事故は記憶に新しいところです．せっかくの機能安全も，不正に改造されてしまえばひとたまりもありません．前述のような処理技法と同時に，不正改造に対する防御策も設計しておくことが大切です．このような技術を耐タンパ（tamper）技術といいます．完ぺきな方法というのは存在しないので，複数の方法を投入可能な範囲で実装しておくことが有効となります．

内容の性質上，対策法の詳細を記述するわけにはいきま

COLUMN　安全に対する意識は日々の会話から

以前，某セット・メーカの偉い方から，「うちの製品は設計ミスをすると人が死にますから」と言われたことがあります．背筋の伸びる思いでした．設計専門の会社で仕事をしていると，どうしてもバーチャルなところに意識を置いてしまいがちです．このときの言葉は，その後の品質に関する考え方に大きく影響しています．今でも氏には感謝しています．

設計専門会社で働いている技術者の方には特に，積極的にセット・メーカの担当者と意見交換することをお薦めします．またセット・メーカの技術者の方は，外注先の若者にもぜひ怖い話をたくさんしていただきたいと思います．いろいろな規定，手法，技法はあれど，設計は人間が行います．安全に対してどういう意識を持つかというのが，実は最も重要なのではないでしょうか．

せんが，代表的な考え方を三つほど紹介します．

1）暗号化

フラッシュROMの普及により，組み込みソフトウェアでも市場でプログラムを書き換えるケースが増えています．暗号化されていない場合，逆アセンブルなどの手法により容易に改造を許してしまいます．

2）チェック・ルーチンの分散

改造に対するチェックを入れても，実際にチェック結果を検査する個所が1カ所だけであれば，それをバイパスさせることは容易となります．

3）ソース公開管理

意外な盲点がこれです．信頼性と安全性向上のためにソースをオープンにすることは重要です．しかし，限られた範囲であれ，ソース・プログラムという最も情報量の大きいものの一つを流出させてしまう危険も内在しています．汎用処理など複数のシステムに適用されているソフトウェアの情報は，それを開示することで第三者のシステム情報の一部の漏えいにもつながることを認識しておく必要があります．

また，レビューにおいて第三者を参画させることは品質上意義深いことですが，安全性の観点から見れば危険を伴います．システムあるいは構成部品の安全性への重要度に応じて，適切に情報を管理することが大切です．

参考・引用*文献

(1) IEC 61508, "Functional Safety of Electrical / Electronic / Programmable Electronic Safety Related Systems", Part1, 2, 4, 5, 1998, Part2, 6, 7, 2000.

たち・のぶゆき
NECマイクロシステム（株）

<筆者プロフィール>
1983年入社．デバイス会社でソフト一筋．経歴の長さにスキルが比例しないのが悩ましい．

3